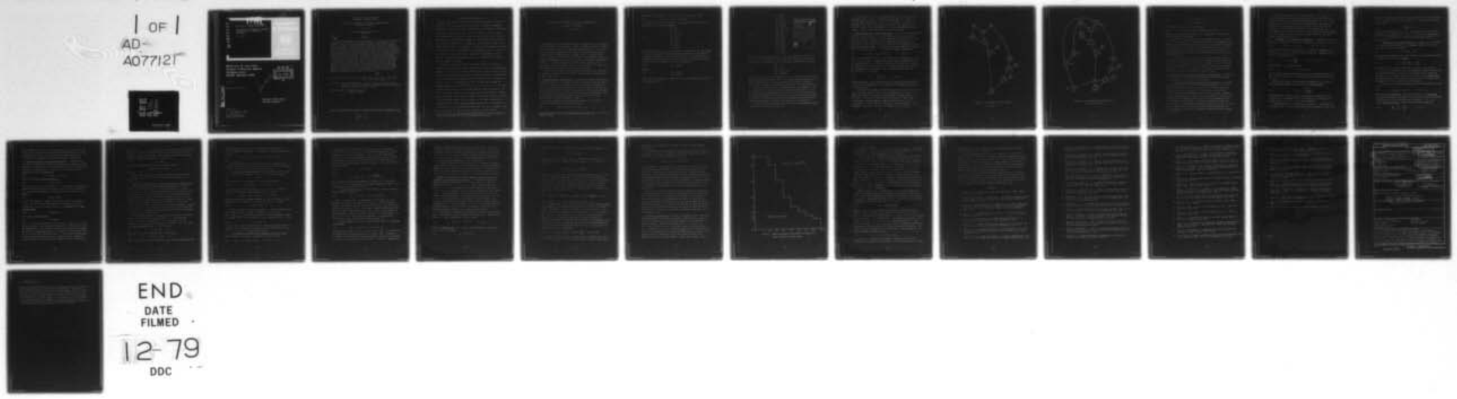
AD-A077 121 WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER F/G 12/1
APPLICATIONS OF SOFTWARE FOR AUTOMATIC DIFFERENTIATION IN NUMER--ETC(U)
JUL 79 L B RALL DAAG29-75-C-0024
UNCLASSIFIED MRC-TSR-1976 NL

| OF |
AD-
A077121



END
DATE
FILMED
12-79
DDC

MRC Technical Summary Report #1976

APPLICATIONS OF SOFTWARE FOR AUTOMATIC DIFFERENTIATION IN NUMERICAL COMPUTATION

L. B. Rall

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

See 1473

July 1979

(Received June 18, 1979)

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

APPLICATIONS OF SOFTWARE FOR AUTOMATIC DIFFERENTIATION IN NUMERICAL COMPUTATION

L. B. Rall

Technical Summary Report #1976
July 1979

ABSTRACT

At present, software for formula translation is used routinely in numerical computation. On the other hand, although software for differentiation of formulas is easy to construct and widely available, many numerical analysts seem to be unaware of its existence and potential for numerical computation. A simple procedure for formula translation and differentiation will be described, and some significant applications will be indicated. In ordinary computation, these include solution of ordinary and partial differential equations by series methods (Taylor and Lie series, for example), solution of nonlinear systems of equations by Newton's method and its variants, and nonlinear optimization (constrained and unconstrained). Together with interval analysis, differentiation can be used to determine properties of functions and thus automate the application of certain theorems, such as ones for existence of fixed points of n-dimensional operators or solutions of nonlinear systems of equations. Another large field of application of differentiation is to automatic error analysis, either using the graph structure of the computation, or interval analysis. An example is given of a program for numerical integration in which automatic differentiation and interval analysis are combined to provide _a priori_ and _a posteriori_ error bounds for the results.

AMS (MOS) Subject Classifications: 65D30, 65G05, 65G10, 65H10, 65K10, 65L05, 65V05, 68-04, 68C05, 68C20

Key Words: Automatic differentiation, Taylor coefficient generation, Error estimation, Differential equations, Nonlinear systems, Optimization, Numerical integration, Interval analysis

Work Unit Numbers 7 (Numerical Analysis)
8 (Computer Science)

Research sponsored by the United States Army under Contract No. DAAG29-75-C-0024.

P-1

SIGNIFICANCE AND EXPLANATION

Money can be saved by turning routine clerical tasks over to electronic computers. Such work includes many of the details of computer programming itself, and a number of high level languages have been introduced to assist with the writing of software. One feature of these languages is formula translation, which prepares subroutines to evaluate functions written in a notation similar to that used in ordinary mathematics. Many scientific and engineering calculations require derivatives of functions or coefficients of power series expansions of functions. The formulation of subroutines to evaluate derivatives of functions or Taylor coefficients can be performed by a computer in much the same manner as formula translation. Software for these purposes has been developed at the Mathematics Research Center over the past 15 years, and has made the computer more useful.

This report describes techniques of automatic differentiation by list processing, and the generation of Taylor coefficients by the use of recurrence relations. A number of possible applications are mentioned, including the solution of differential equations, nonlinear systems of equations, constrained and unconstrained optimization problems, and various methods for error estimation. Some of these have been implemented by actual software, and an example is given of automatic error estimation for a numerical integration. In this latter example, differentiation is used in combination with interval arithmetic. The use of derivatives increases the applicability of interval arithmetic considerably, as explained in this report. However, unlike interval arithmetic, software for differentiation can be written entirely in a high level language such as FORTRAN, and thus is easily portable between installations. As indicated in this report, the use of this software where appropriate can result in savings of time and effort, as well as a more effective approach to certain problems.

APPLICATIONS OF SOFTWARE FOR AUTOMATIC DIFFERENTIATION IN NUMERICAL COMPUTATION

L. B. Rall

1. Introduction. At the foundations of numerical computation lie the means by which calculations are actually done. Today, this means not only the computing machines (hardware), but also the programs (software) available to the user. This paper describes software for differentiation of functions and some of its applications. Differentiation is somewhat similar to washing dishes, a dull, uninteresting task best left to a machine. Two methods are given for automatic differentiation, and their use in various programs for applications is described. As the discussion is limited to actual experience at the Mathematics Research Center over the past 15 years, no mention is made of much fine software which has doubtless been developed elsewhere.

At the end of the paper, some conclusions concerning the past usefulness of automatic differentiation will be given, together with some predictions of possible future trends.

2. Differentiation by list processing. An approach to formula translation and differentiation which is conceptually simple is provided by a list processing technique. Rather than use a general purpose list processing language, however, it was considered more appropriate to develop software for the specific purpose of evaluating and differentiating formulas. This technique, described in the book by Rall [35, pp. 155-160], forms the basis of the program written in 1965 by Reiter [40] and later extended by Gray and Reiter [15] and Wertz [45,46], the latter assisted by T. Ladner.

The principles are best illustrated by an example [35, pp. 155-156]. Suppose that one wishes to evaluate the function

$$f(x,y) = (xy + \sin x + 4)(3y^2 + 6) \tag{2.1}$$

Research sponsored by the United States Army under Contract No. DAAG29-75-C-0024.

and some of its derivatives. The first step is to decode the formula

(2.2) $$F = (X*Y + SIN(X) + 4)*(3*Y**2 + 6)$$

corresponding to (2.1) into a sequence of arithmetic operations and calls to subroutines. This gives the <u>code list</u>

(2.3)
```
T1 = X*Y
T2 = SIN(X)
T3 = T1 + T2
T4 = T3 + 4
T5 = Y**2
T6 = 3*T5
T7 = T6 + 6
F = T4*T7 .
```

Note that the code list is itself a sequence of statements in the same language in which (2.2) is written, and hence can be translated by the same compiler into machine language. Now, to differentiate (2.3), the program refers to a dictionary containing the derivatives of the arithmetic operations and subroutines called. For example, the entry corresponding to

(2.4)
```
U2 = SIN(U1)
```

would be

(2.5)
```
V1 = COS(U1)
DU2 = V1*DU1
```

and so forth. Applying this procedure to (2.3), one obtains the code list for the differential of F,

```
                V1 = X*DY
                V2 = DX*Y
               DT1 = V1 + V2
                V3 = COS(X)
               DT2 = V3*DX
               DT3 = DT1 + DT2
               DT4 = DT3
(2.6)           V4 = Y**1
                V5 = 2*V4
               DT5 = V5*DY
               DT6 = 3*DT5
               DT7 = DT6
                V6 = T4*DT7
                V7 = DT4*
                DF = V6 + V7
```

Accession For	
NTIS GRA&I	✓
DDC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or special
A	

In order to obtain the code list for $\partial f/\partial x$, for example, one sets DX = 1, DY = 0 in (2.6) and obtains, after elimination of trivialities (multiplications by 1 or 0, exponentiation to the first power, addition of 0), the list

```
               DXT1 = Y
               DXT2 = COS(X)
(2.7)
               DXT4 = DXT1 + DXT2
                DXF = DXT4*T7
```

In this case, the list for the derivative DXF is simpler than the list for F, due to the polynomial terms in $f(x,y)$. For nonpolynomial functions, one would expect differentiation to yield more complex lists; however, polynomials are not at all uncommon in the modelling and analysis of nonlinear problems.

The most important feature of differentiation by list processing is that the output (2.7) is a list of exactly the same form as the input list (2.3), and can itself be differentiated with respect to any variable entering into it, simply by invoking the same processor. Also, if one needs only some high derivative of a function, the lists for the intermediate derivatives can be discarded after they are processed, or at least it is not necessary to compile them into machine language. This can result in a considerable saving of computer time and storage locations.

It is instructive to view the procedure described above in terms of the Kantorovich graph [17] which is equivalent to the code list. Figure 2.1 shows the graph corresponding to the list (2.3), Figure 2.2 the graph for computing both F and its partial derivative DXF with respect to X. To obtain a graph for the computation of DXF only, the nodes indicated by squares in Figure 2.2 are unnecessary, and can be eliminated.

3. <u>Automatic generation of Taylor coefficients</u>. This is the approach adopted by Moore et al. in 1964 [30] in connection with the solution of ordinary differential equations by series, and was implemented by Reiter in 1965 [38] and 1967 [41] as an independent program. Recursive generation of Taylor coefficients is also incorporated in the more general program differentiation software developed by Kedem [18], a system which is also capable of supervising the list processing approach. To illustrate this method, the notation

$$(3.1) \qquad f_0 = f(x_0), \quad f_\tau = \frac{1}{\tau!} f^{(\tau)}(x_0), \qquad \tau = 1,2,\ldots,j\ ,$$

will be used for the first $j + 1$ Taylor coefficients of the function $f(x)$ at $x = x_0$. (In case f depends on several variables, the derivatives in (3.1) are understood to be partial with respect to the variable of interest.) The recursion relations for generating the successive Taylor coefficients for the arithmetic operations and various functions are well known [27, pp. 107-118; 19]. For example,

$$(3.2) \qquad (fg)_j = \sum_{\tau=0}^{j} f_\tau g_{j-\tau}, \qquad j = 0,1,2,\ldots\ .$$

These relationships can be used to process the code list for the evaluation of a function in the same way as the formulas for differentials were used in the previous section.

To interpret this method in terms of the Kantorovich graph, suppose that instead of simply values of terms, a vector consisting of the first $j + 1$ Taylor coefficients is received by each node along the edges coming into it from above. This vector is then processed at the node by invoking the corresponding recurrence relations, and the resulting vector is then transmitted to the lower nodes connected to it. Using the graph in Figure 2.1, for example, one obtains the Taylor coefficients (3.1) by setting

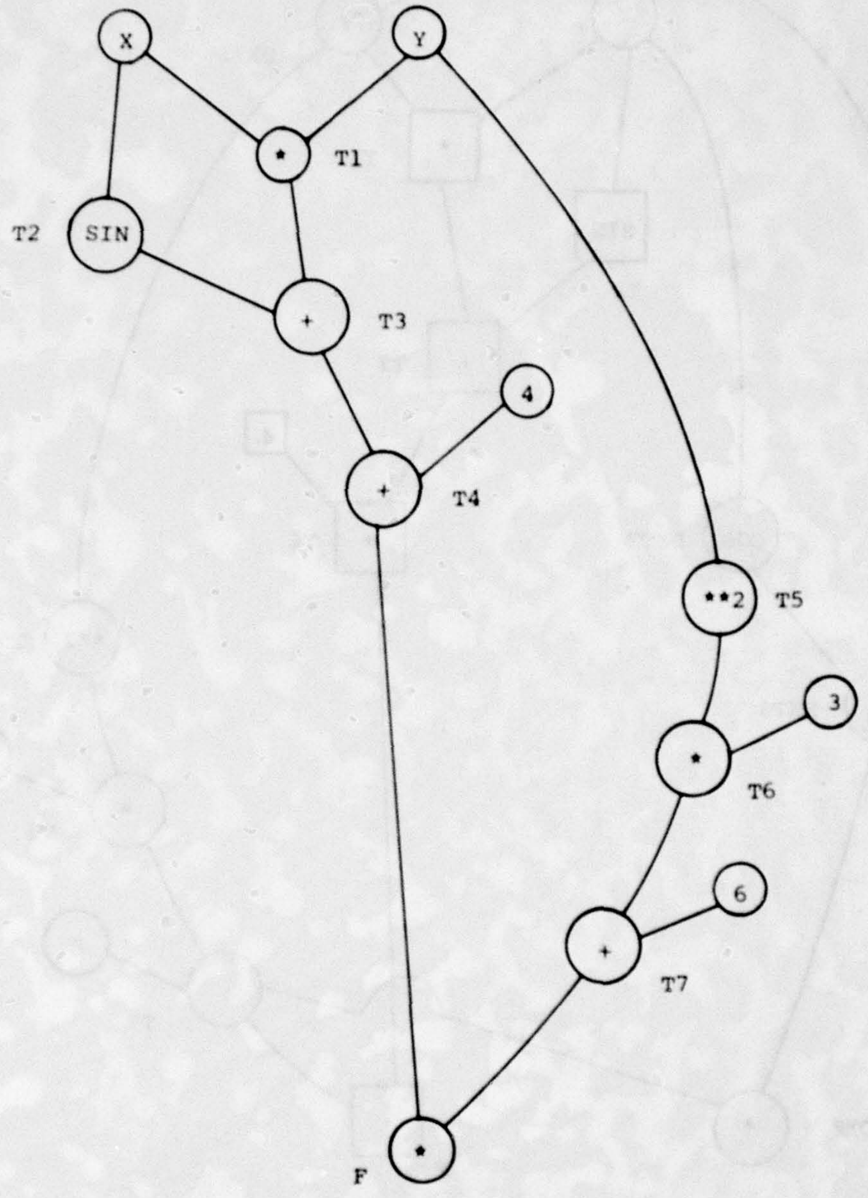


Figure 2.1. The graph of the calculation of f(x,y) .

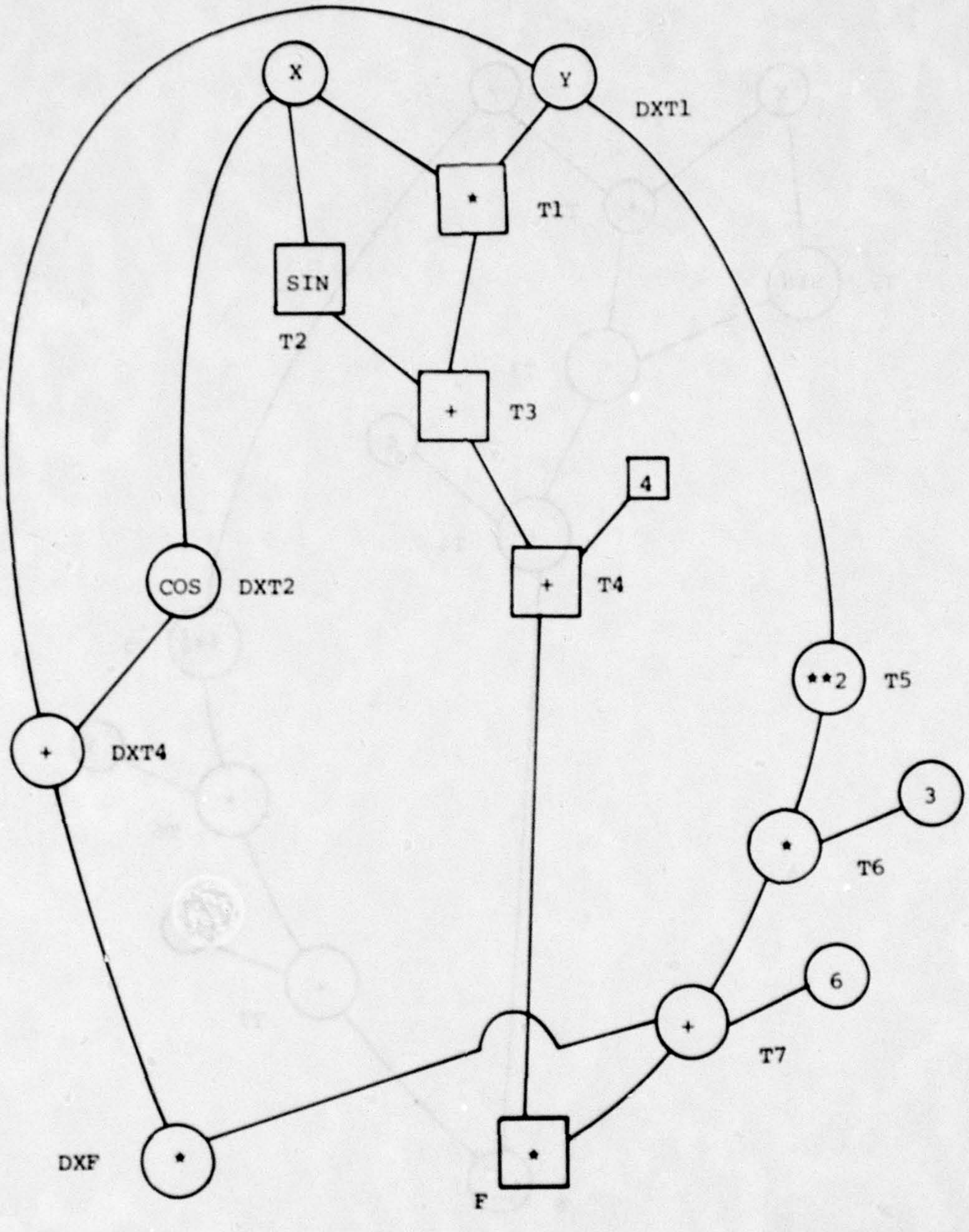


Figure 2.2. The graph of the calculation of $f(x,y)$ and $f_x(x,y)$.

$$(3.3) \quad \begin{cases} X = (x_0, 1, 0, \ldots, 0), \\ Y = (y_0, 0, 0, \ldots, 0). \end{cases}$$

At the node labelled T3, one would compute

$$(3.4) \quad T3(TAU) = T1(TAU) + T2(TAU), \quad TAU = 0(1)J,$$

and so on. The Taylor coefficients of f considered as a function of y may be obtained in the same way.

This method of differentiation has the advantage that only the evaluation list (2.3) is needed. It is not possible to discard intermediate values when only higher derivatives are wanted, and the list processing approach is better adaptable to the computation of mixed partial derivatives. However, if Taylor series expansions with respect to one variable or only first partial derivatives are required, then the method described in this section is highly effective.

It is one thing to describe methods for automatic differentiation in a general way as done here, but quite another to produce quality software for this purpose, as has been done by Reiter, Gray, Wertz, and others cited in this paper and elsewhere. Of course, although this software has performed satisfactorily for a number of years, there is always the possibility of improvement by making use of advances in computer science.

4. Some traditional applications of automatic differentiation. A number of possibilities for the use of software for automatic differentiation come to mind immediately. A few will be sketched here.

a. Solution of differential equations by series methods. The software developed by Moore et al. in 1964 [30] and Braun and Moore in 1967 [3] is based on the methods for Taylor series solution of ordinary differential equations described in the papers [25,26] and books [27,28] by Moore. Although these programs were designed to be used with interval arithmetic to provide automatic error estimates, they can be executed in ordinary real (floating-point) arithmetic to obtain results which are competitive with other methods of the same order. In particular, Moore [28] refutes the statement made in standard texts on numerical analysis [4, p. 214; 5, p. 330] that "the necessity of calculating the higher derivatives makes Taylor's algorithm completely unsuitable on high-speed computers for general integration purposes."

The related technique of the Lie-series method for the numerical solution of differential equations also makes use of the automatic generation of Taylor coefficients, as described by Knapp and Wanner [19]. The original program [20], written in 1968, required the user to write the sequence of subroutine calls corresponding to a code list of the form (2.3) for the integrand. In 1969, automatic generation of the code list was added by T. Szymanski, under the direction of Julia Gray, to obtain an improved program [21]. Once again, the result is apparently competitive with Runge-Kutta methods of the same order [19,37].

b. Solution of nonlinear systems of equations. A popular and effective method for the solution of systems of n nonlinear equations in n unknowns,

$$(4.1) \qquad f_i(x_1, x_2, \ldots, x_n) = 0, \qquad i = 1,2,\ldots,n \ ,$$

is the Newton-Kantorovich algorithm [16,35]. However, to implement this method in such a way as to obtain quadratic convergence, one must calculate the n^2 partial derivatives

$$(4.2) \qquad f'_{ij} = \frac{\partial f_i}{\partial x_j}, \qquad i,j = 1,2,\ldots,n \ ,$$

to obtain the $n \times n$ Jacobian matrix

$$(4.3) \qquad J = f'(x) = (f'_{ij}) \ .$$

This is done by automatic differentiation in the 1967 program of Gray and Rall [10] and the 1972 version prepared by Kuba and Rall [23] and described in [11,35].

Higher order methods, such as Chebyshev's method and the inversion of power series would require computation of the Hessian operator

$$(4.4) \qquad H = f''(x) = \left(\frac{\partial^2 f_i}{\partial x_j \partial x_k} \right)$$

and perhaps higher derivatives [35]. The automatic formation of (4.4) is implemented in the programs [10,23] in connection with verification of existence of solutions of (4.1) and error estimation.

Another approach to the solution of the system (4.1) using derivatives is a continuation or homotopy method based on the conversion of (4.1) into a system of differential equations for a curve connecting an initial approximation x_0 for which $f(x_0) = y$ to a solution x^*. Such a system is, using vector notation

$$(4.5) \qquad J \frac{dx}{dt} = -y ,$$

with the initial conditions $x(0) = x_0$. Integrating (4.5) by some curve-following technique [24] from $t = 0$ to $t = 1$ will give an approximation to $x(1) = x^*$. The system (4.5) can be constructed using software for differentiation.

c. Optimization of nonlinear functionals. A problem arising frequently in the applications of mathematics is to optimize (maximize or minimize) the value of a nonlinear functional

$$(4.6) \qquad g = g(x_1, x_2, \ldots, x_n) .$$

With sufficient smoothness, this can be converted into the problem of solving the system (4.1) by taking the gradient

$$(4.7) \qquad \nabla g = \left(\frac{\partial g}{\partial x_1}, \frac{\partial g}{\partial x_2}, \ldots, \frac{\partial g}{\partial x_n} \right)$$

and setting $\nabla g = 0$, which is (4.1) with $f_i(x_1, x_2, \ldots, x_n) = \partial g/\partial x_i$, $i = 1,2,\ldots,n$. If one wants to solve this system by a method of Newton type, then the Hessian matrix $(\partial^2 g/\partial x_i \partial x_j)$ is required, which is simply the Jacobian (4.3). In problems of this type, which are called unconstrained optimization problems, the formation of the gradient and the Hessian can be automated by the use of software for differentiation.

Another type of optimization problem requires the satisfaction of conditions

$$(4.8) \qquad h_\tau(x_1, x_2, \ldots, x_n) = 0, \qquad \tau = 1,2,\ldots,k ,$$

which are called constraints. Given enough smoothness, the constrained optimization problem can be reduced to the solution of a system of equations of the form (4.1) by the introduction of new unknowns $\lambda_1, \lambda_2, \ldots, \lambda_k$, called Lagrange multipliers. One obtains the nonlinear equations

$$(4.9) \qquad \frac{\partial g}{\partial x_i} + \sum_{\tau=1}^{k} \lambda_\tau \frac{\partial h_\tau}{\partial x_i} = 0, \qquad i = 1,2,\ldots,n ,$$

which, together with (4.8), form a system of order $n + k$ to solve. The equations (4.9) can be constructed by automatic differentiation as in the unconstrained case, and, if desired, the entire system can itself be differentiated to implement its solution by a method of Newton type.

5. Analysis of roundoff error using differentiation. A nontraditional application of software for automatic differentiation in numerical analysis would be the implementation of the method of Bauer [1] for the study of the propagation of roundoff error. This technique is based on the use of relative (logarithmic) differentials,

$$\rho f = df/f, \quad \rho x = dx/x \ , \tag{5.1}$$

and the corresponding relative derivatives

$$\rho f/\rho x = xf'/f \ , \tag{5.2}$$

and the code list or Kantorovich graph for f. The quantities needed for the analysis of roundoff error in the calculation of the function defined by (2.2) are available immediately from the code lists (2.3) and (2.4), for example,

$$\text{RHOT5} = \text{DT5/T5} \ . \tag{5.3}$$

Setting DX = X•RHOX, DY = Y•RHOY would lead to a code list expressed entirely in terms of relative differentials. According to Bauer [1], a rounding error in the calculation of say, T4, is harmless if the input condition number

$$|\text{RHOT4/RHOX}| \geq 1 \ , \tag{5.4}$$

or the output condition number

$$|\text{RHOF/RHOT4}| \leq 1 \ . \tag{5.5}$$

Here, F is being considered to be a function of X only, with Y fixed. Similar conditions can be formulated for functions of several variables [1]. The significance of (5.4) is that a roundoff error in T4 does not exceed the result of a corresponding error in X, while (5.5) means that an error in T4 will not affect the final value of F by more than the same amount. An application of these ideas would be to examine final or intermediate condition numbers of several alternative graphs (or lists) for calculating the same function in order to pick one in which roundoff error is the least malignant. In order to automate this process, one can use the existing software or, perhaps better, write a list processor which produces relative rather than ordinary differentials. For example, for

(5.6) $$U3 = U1 \cdot U2\ ,$$

the dictionary entry for the relative differential would be

(5.7) $$RHOU3 = RHOU1 + RHOU2\ ,$$

and so on.

Another direct application of automatic differentiation to roundoff error analysis would be to automate the linearization technique due to Stummel [44].

6. Computational verification of existence of solutions of systems of equations using interval analysis and automatic differentiation. Interval analysis [27,28] provides a method for obtaining bounds for the ranges of computable functions, including perturbations due to roundoff error in the actual computations. Software for computing with interval arithmetic was developed side-by-side with differentiation software by Reiter [39,42], starting in 1965. A modern package has been prepared by Yohe [47]. Unlike the programs for differentiation, which can be written in a high level language, interval software was originally highly machine dependent, including input and output [2]. Using advances in computer science, Crary and Ladner [9], Crary [6,7,8], and Yohe [48,49] have been able to reduce this dependence to a minimum, at the cost of speed.

For the present purposes, it is adequate to know that one can extend numerical computations from ordinary numbers x to closed intervals $X = [a,b]$ (which include numbers if $a = b$). Execution in interval arithmetic of a program for the calculation of a function $f(x)$ will result in an interval extension $F(X)$ of $f(x)$ such that

(6.1) $$f(x) \in F(X), \quad x \in X\ .$$

Thus, for $|X| = |[a,b]| = \max\{|a|,|b|\}$, one will have

(6.2) $$|f(x)| \le |F(X)|, \quad x \in X\ .$$

These results extend to finite dimensional spaces, using the maximum absolute component norm for vectors, and the corresponding norm for matrices. In R^n_∞, the closed ball with center y and radius ρ can be represented as an interval,

(6.3) $$\bar{U}(y,\rho) = \{x : \|x - y\|_\infty \leq \rho\} = [y - \rho e, y + \rho e] ,$$

where $e = (1,1,\ldots,1)$.

An immediate application of these techniques is to automate some theorems from classical and interval analysis on the existence of solutions of systems of equations (4.1) and to obtain error bounds.

a. <u>The contraction mapping theorem</u>. Here, one transforms the system (4.1), written in vector form as $f(x) = 0$, as a fixed point problem

(6.4) $$x = \phi(x)$$

for some operator ϕ, for example, to use an iteration method

(6.5) $$x_{n+1} = \phi(x_n), \qquad n = 0,1,2,\ldots ,$$

starting from some initial approximation $x_0 = y$. If ϕ is Lipschitz continuous in some sufficiently large ball $\bar{U}(y,\rho)$, that is,

(6.6) $$\|\phi(x) - \phi(z)\| \leq \alpha\|x - z\|, \qquad x,z \in \bar{U}(y,\rho) ,$$

then it follows from the theorem of Banach [35, pp. 64-74] that the sequence generated by (6.5) converges to a solution $x^* \in \bar{U}(y,\rho)$, provided

(6.7) $$\rho \geq \|x_1 - x_0\|/(1 - \alpha) .$$

In order to obtain the Lipschitz constant α, automatic differentiation may be used to compile a program to evaluate the $n \times n$ matrix $\phi'(x) = (\partial\phi_i/\partial x_j)$. Using interval arithmetic, the same program will compute the extension $\phi'(X)$ on the interval $X = \bar{U}(y,\rho)$ defined by (6.3). If then

(6.8) $$\alpha = |\phi'(X)| < 1$$

and (6.7) holds, then the hypotheses of the contraction mapping theorem are satisfied. This computation, if successful, also yields the error bound $\|y - x^*\| \leq \rho$ for y as an approximate solution.

b. <u>The Kantorovich theorem on the convergence of Newton's method</u>. Newton's method, as applied to the system (4.1), may be written

(6.9) $$x_{n+1} = x_n - [f'(x_n)]^{-1} f(x_n), \qquad n = 0,1,2,\ldots, \qquad x_0 = y ,$$

in vector form. The theorem of Kantorovich [16] requires the numbers $B_0 \geq \|[f'(y)]^{-1}\|$, $y_0 \geq \|x_1 - x_0\|$, both obtainable rigorously by interval computation, and a Lipschitz constant K for f' on some sufficiently large ball $X = \bar{U}(y,\rho)$. The programs described in [10,11,23] use automatic differentiation to obtain the bilinear operator $f''(x)$ defined by (4.4). Evaluation by interval arithmetic gives the extension $F''(X)$ and the desired Lipschitz constant

$$K = |F''(X)| . \tag{6.10}$$

Now, if

$$h_0 = B_0\eta_0 K \leq \frac{1}{2}, \quad \rho \geq \frac{1 - \sqrt{1 - 2h}}{h}\eta_0 , \tag{6.11}$$

then the Kantorovich theorem guarantees the existence of x^* such that $f(x^*) = 0$ in X, the convergence of the sequence generated by (6.9) to x^*, and provides the error bound $\|y - x^*\| \leq \rho$.

c. An interval Newton's method. One may obtain an interval version of Newton's method by setting

$$X_{n+1} = X_n \cap \{m(X_n) - [F'(X_n)]^{-1}F(m(X_n))\} , \tag{6.12}$$

$n = 0,1,2,\ldots$, where $m(X_n)$ is the midpoint of the interval X_n, as described by Nickel [31] and Moore [27]. Here, X_0 can be an arbitrary interval, not necessarily a ball, and the condition $X_{n+1} \subset X_n$ implies the existence of a solution $x^* \in X_n$. Once again, automatic differentiation and interval arithmetic are used to obtain $F'(X_n)$, which is then inverted by interval methods. Inversion of interval matrices is a tedious process, however, so (6.12) is of little practical importance. This computation is available as an option in the program of Kuba and Rall [23].

d. Moore's theorem. This is an interval theorem, based on the Krawczyk transformation

$$K(X) = y - Yf(y) + \{I - YF'(X)\}(X - y) \tag{6.13}$$

of an arbitrary interval X [22]. In (6.13), $y \in X$ and Y is an arbitrary, nonsingular real (not interval) matrix. Once again, $K(X)$ is computed by automatic differentiation and interval arithmetic, and has been implemented as an option in the program [23] by Julia Gray. If $K(X) \subset X$, then Moore's theorem [29] guarantees the existence of a solution $x^* \in X$, which, as in the previous method, also yields error bounds for $x^* - m(X)$ or any other approximate solution. Rall [36] has shown that the Kantorovich theorem has little theoretical advantage over Moore's theorem, while the latter is much less costly to apply. Hence, the elaborate implementation of the calculation of (6.10) in the programs [11,23] can be discarded.

It should be pointed out that the purpose of the methods and programs described in this section is not to solve equations, but rather to guarantee existence of solutions and give rigorous error bounds for approximate solutions. This type of computation is expensive, but may be justifiable in certain applications, such as the design of critical components of aircraft.

7. Automation of estimation of error of numerical integration and other techniques of classical numerical analysis. In ordinary applications of numerical analysis, error estimation is a tedious chore which should be mechanized as much as possible. As errors in data, roundoff error, and truncation error contaminate almost every actual computation, some analysis of their effects is required in order to judge the reliability of the results obtained. Interval arithmetic is suitable for keeping track of data and roundoff errors automatically. If expressions for the truncation error are known in terms of derivatives, as in classical numerical analysis, then automatic differentiation can be used in connection with interval arithmetic to obtain bounds for truncation error. All of this is illustrated aptly by the work of Moore on ordinary differential equations [3,25,26,27,28,30], and by the following examples from ordinary numerical analysis.

The processes of interpolation and numerical integration and differentiation can be regarded as coming from some linear functional Λ applied to the function f under consideration. Thus, one writes

$$\text{(7.1)} \qquad \Lambda f = \sum_{i=1}^{n} \alpha_i f(x_i) + \left(\frac{1}{n}\right)^p t(\xi), \qquad \xi \in [a,b] ,$$

where the remainder term $t(\xi)$ is some linear combination of derivatives of f at ξ, for example,

$$\text{(7.2)} \qquad t(\xi) = C_p f^{(p-1)}(\xi) ,$$

where the constant C_p is known. The linear combination of values of f,

$$(7.3) \qquad Rf = \sum_{i=1}^{n} \alpha_i f(x_i)$$

is usually called the rule of numerical interpolation, integration, or differentiation, etc. Taking an interval extension of (7.1) gives, for $X = [a,b]$,

$$(7.4) \qquad \Lambda f \in \sum_{i=1}^{n} A_i F(x_i) + \overline{\left(\frac{1}{n}\right)^p} T(X) ,$$

where the interval on the right can be computed automatically by use of the software described in this paper. Thus, this furnishes a complete error analysis of the use of the rule Rf given by (7.3) in order to calculate the functional Λf. This type of analysis is carried out in the programs of Gray and Rall [12,13,14] for bounding the error of various rules for the numerical integration of functions of a single variable. For example, for Simpson's rule, the form of (7.4) is

$$(7.5) \qquad \int_a^b f(x)dx \in \frac{w[a,b]}{6}\left[Y(A) + 4Y\left(\frac{A+B}{2}\right) + Y(B)\right] - \frac{(w[a,b])^5}{2880} Y^{iv}([a,b])$$

where $w[a,b] = b - a$ is the width of the interval $[a,b]$.

Although the speed of the computer blurs the distinction between a priori and a posteriori error estimations, an expression of the form can furnish either. For example, (7.4) can be computed for a small value of n, and then its width can be extrapolated to larger values of n, as the width of T(X) from the original computation will always be an upper bound for $T(X_n)$, $X_n \subset X$. This method for optimization of time or accuracy in numerical integration is implemented in the program [13]. A posteriori error estimates, of course, are always available directly from the computation of (7.4). Once it has been established that $\Lambda f \in [c,d]$, one may take as an estimate for Λf one of the numbers

$$(7.6) \qquad m[a,b] = \frac{a+b}{2}, \quad h[a,b] = \frac{2ab}{a+b}, \quad g[a,b] = \sqrt{ab} ,$$

which minimize the absolute error, relative error, or maximizes the relative precision in the sense of Olver [32], respectively. Interval results can be intersected, if more than one is available, to obtain a possible improvement [12,13].

As an example of the application of this program, the region to the left and below of the staircase in Figure 7.1 shows where 5-place accuracy for the calculation of the elliptic integral of first kind

$$(7.7) \qquad F(k,\theta) = \int_0^{\varphi} (1 - k^2 \sin^2\theta)^{-\frac{1}{2}} d\theta$$

can be guaranteed, using Simpson's rule with 7 nodes (three times on $[0,\varphi]$). This guarantee includes roundoff on the UNIVAC 1110 in single precision (to about 9 decimal digits), and should be valid for calculators carrying more places. A result of this kind might be useful in the design of software for programmable hand calculators, however, the important point is that the results shown in Figure 7.1 were obtained interactively at a computer terminal. The user supplied the integrand of (7.7), and then various values of k and φ. The detailed error analysis was carried out by the computer, using the program [13].

The error of numerical differentiation can be analyzed in the same way. It may seem strange to use an analytic differentiator for this purpose; however, it may turn out to be cheaper or more suitable to the input data to calculate derivatives as linear combinations of function values than to execute the differentiation subroutine, once the rule for numerical analysis has been established to be of sufficient accuracy for the functions considered. Here again, the elaborate differentiation and interval software is used for error analysis in the design of a program for production computation, not for the computation itself.

8. Conclusions and predictions.

The above illustrations give an idea of the power and usefulness of software for automatic differentiation, however, in spite of its portability, it is not widely accepted at the present time by numerical analysts. One hears objections such as: (i) It won't work; (ii) it won't work well; or (iii) a problem is known to which it does not apply. Objections (i) and (ii), coming from people who use formula translators routinely, are demolished by the facts that differentiators are based on the same principles as translators and have been used with success at the Mathematics Research Center and elsewhere for more than a decade. Objection

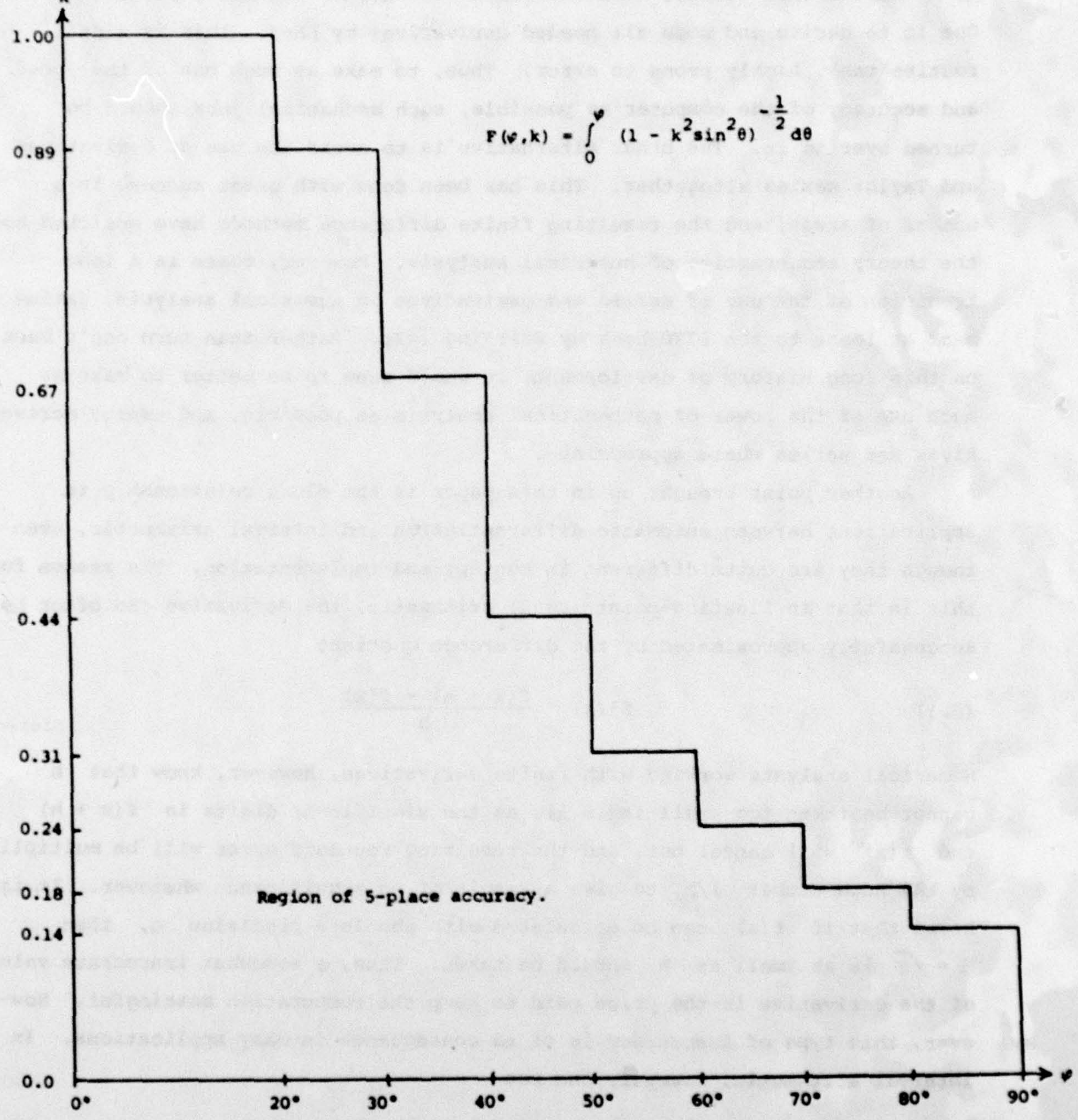


Figure 7.1. Region of guaranteed 5-place accuracy for calculation of $F(\varphi,k)$ by Simpson's rule with 7 nodes.

(iii) is simply a negative, nonconstructive remark which applies to any method or tool. A more positive approach would be to think of problems to which this software does apply, as the examples given above indicate.

There are, of course, alternatives to the use of automatic differentiators. One is to derive and code all needed derivatives by hand. This is a dull, routine task, highly prone to error. Thus, to make as much use of the speed and accuracy of the computer as possible, such mechanical jobs should be turned over to it. The other alternative is to avoid the use of derivatives and Taylor series altogether. This has been done with great success in a number of areas, and the resulting finite difference methods have enriched both the theory and practice of numerical analysis. However, there is a long tradition of the use of series and derivatives in numerical analysis, dating back at least to the 1730 book by Stirling [43]. Rather than turn one's back on this long history of development, it would seem to be better to make as much use of the power of mathematical analysis as possible, and employ derivatives and series where appropriate.

Another point brought up in this paper is the close relationship in applications between automatic differentiation and interval arithmetic, even though they are quite different in concept and implementation. The reason for this is that in floating-point (real) arithmetic, the derivative can often be successfully approximated by the difference quotient

$$f'(x) \sim \frac{f(x+h) - f(x)}{h} . \tag{8.1}$$

Numerical analysts working with finite derivatives, however, know that h cannot be taken too small in (8.1), as the significant digits in $f(x+h)$ and $f(x)$ will cancel out, and the resulting roundoff error will be multiplied by the huge number $1/h$ to give a result of no significance whatever. It is known that if $f(x)$ can be calculated with absolute precision η, then $h = \sqrt{\eta}$ is as small as h should be taken. Thus, a somewhat inaccurate value of the derivative is the price paid to keep the computation meaningful. However, this type of inaccuracy is of no consequence in many applications. In interval arithmetic, however, one has

$$[0,1] - [0,1] = [-1,1] , \tag{8.2}$$

so cancellation is replaced by a widening of intervals, which is only aggravated by multiplying by large numbers. This "painful honesty" of interval arithmetic makes it more suitable for use with derivatives than difference quotient approximations such as (8.1).

From an historical standpoint, formula translators did not come into general use until hardware for floating-point arithmetic became widely available. Then, one could declare a variable to be "real", and forget the scaling, etc. which is better done in machine or assembly language for fixed-point computations. Because of the intimate relationship between interval analysis and derivatives, one may have to wait for hardware for *interval arithmetic* before differentiation by software is widely accepted.

Finally, the present breed of automatic differentiators, as all human products, can be improved. At present, this seems to be a task which is too exotic for numerical analysts and too mundane for computer scientists. However, as with formula translators, one can expect improvements to come with use and coupled with advances in hardware and computer science.

REFERENCES

1. Bauer, F. L.: Computational graphs and rounding error, SIAM J. Numer. Anal. 11 (1974), 87-96.

2. Binstock, W., Hawkes, J., and Hsu, N.-T.: An interval input/output package for the UNIVAC 1108, Tech. Sum. Rpt. No. 1212, Math. Res. Center, Univ. Wisconsin-Madison, Sept., 1973.

3. Braun, J. A. and Moore, R. E.: A program for the solution of differential equations using interval arithmetic (DIFEQ) for the CDC 3600 and 1604, Tech. Sum. Rpt. No. 901, Math. Res. Center, Univ. Wisconsin-Madison, June, 1968.

4. Conte, S. D.: Elementary Numerical Analysis: An Algorithmic Approach, New York: McGraw-Hill, 1965.

5. Conte, S. D. and de Boor, Carl: Elementary Numerical Analysis: An Algorithmic Approach, 2d Ed., New York: McGraw-Hill, 1972.

6. Crary, F. D.: Language extensions and precompilers, Tech. Sum. Rpt. No. 1319, Math. Res. Center, Univ. Wisconsin-Madison, Feb., 1973.

7. Crary, F. D.: The AUGMENT precompiler. I. User information. Tech. Sum. Rpt. No. 1469, Math. Res. Center, Univ. Wisconsin-Madison, Dec., 1974, revised Apr., 1976.

8. Crary, F. D.: The AUGMENT precompiler. II. Technical documentation, Tech. Sum. Rpt. No. 1470, Math. Res. Center, Univ. Wisconsin-Madison, Oct., 1975.

9. Crary, F. D. and Ladner, T. D.: A simple method of adding a new data type to FORTRAN, Tech. Sum. Rpt. No. 1605, Math. Res. Center, Univ. Wisconsin-Madison, May, 1970.

10. Gray, Julia H. and Rall, L. B.: NEWTON: A general purpose program for solving nonlinear systems, Tech. Sum. Rpt. No. 790, Math. Res. Center, Univ. Wisconsin-Madison, Sept., 1967.

11. Gray, Julia H. and Rall, L. B.: NEWTON: A general purpose program for solving nonlinear systems, Proc. 1967 Army Numer. Anal. Conf., pp. 11-59. Durham, N.C.: Army Res: Office, 1967.

12. Gray, Julia H. and Rall, L. B.: A computational system for numerical integration with rigorous error estimation, Proc. 1974 Army Numer. Anal. Conf., pp. 341-355. Durham, N.C.: Army Res. Office, 1974.

13. Gray, Julia H. and Rall, L. B.: INTE: A UNIVAC 1108/1110 program for numerical integration with rigorous error estimation, Tech. Sum. Rpt. No. 1428, Math. Res. Center, Univ. Wisconsin-Madison, Oct., 1975.

14. Gray, Julia H. and Rall, L. B.: Automatic Euler-Maclaurin integration, Proc. 1976 Army Numer. Anal. and Comp. Conf., pp. 431-444. Research Triangle Park, N.C.: Army Res. Office, 1976.

15. Gray, Julia H. and Reiter, Allen: Compiler of differentiable expressions (CODEX) for the CDC 3600, Tech. Sum. Rpt. No. 791, Math. Res. Center, Univ. Wisconsin-Madison, Dec., 1967.

16. Kantorovich, L. V.: Functional analysis and applied mathematics, Uspehi Mat. Nauk 3 (1948), 89-185 (Russian). Tr. by C. D. Benster, Natl. Bur. Std. Rpt. No. 1509, Washington, 1952.

17. Kantorovich, L. V.: On a mathematical symbolism convenient for performing machine calculations, Dokl. Acad. Nauk SSR 113 (1957), 738-741 (Russian).

18. Kedem, G.: Automatic differentiation of computer programs, Tech. Sum. Rpt. No. 1697, Math. Res. Center, Univ. Wisconsin-Madison, Nov., 1976.

19. Knapp, H. and Wanner, G.: Numerical solution of ordinary differential equations by Groebner's method of Lie-series, Tech. Sum. Rpt. No. 880, Math. Res. Center, Univ. Wisconsin-Madison, June, 1968.

20. Knapp, H. and Wanner, G.: LIESE: A program for ordinary differential equations using Lie-series, Tech. Sum. Rpt. No. 881, Math. Res. Center, Univ. Wisconsin-Madison, June, 1968.

21. Knapp, H. and Wanner, G.: LIESE II: A program for ordinary differential equations using Lie-series, Tech. Sum. Rpt. No. 1008, Math. Res. Center, Univ. Wisconsin-Madison, Aug., 1969.

22. Krawczyk, R.: Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken, Computing 4 (1969), 187-201.

23. Kuba, Dennis and Rall, L. B.: A UNIVAC 1108 program for obtaining rigorous error estimates for approximate solutions of systems of equations, Tech. Sum. Rpt. No. 1168, Math. Res. Center, Univ. Wisconsin-Madison, Jan., 1972.

24. Li, T. Y. and Yorke, J. A.: A simple, reliable numerical algorithm for following homotopy paths (to appear), Proc. MRC Symp. on Analysis and Computation of Fixed Points, New York: Academic Press, 1979.

25. Moore, R. E.: The automatic analysis and control of error in digital computation based on the use of interval numbers, [33], pp. 61-130, 1965.

26. Moore, R. E.: Automatic local coordinate transformations to reduce the growth of error bounds in interval computation of solutions of ordinary differential equations, [34], pp. 103-140, 1965.

27. Moore, R. E.: Interval Analysis. Englewood Cliffs, N.J.: Prentice-Hall, 1966.

28. Moore, R. E.: Methods and Applications of Interval Analysis. Philadelphia: SIAM Publications, 1979.

29. Moore, R. E.: A test for existence of solutions to nonlinear systems, SIAM J. Numer. Anal. 14 (1977), 611-615.

30. Moore, R. E., Davison, J. A., Jaschke, H. R., and Shayer, S.: DIFEQ integration routine - user's manual, Tech. Rpt. LMSC 6-90-64-6, Lockheed Missiles and Space Co., Palo Alto, Calif., 1964.

31. Nickel, K.: On the Newton method in interval analysis, Tech. Sum. Rpt. No. 1136, Math. Res. Center, Univ. Wisconsin-Madison, Dec., 1971.

32. Olver, F. W. J.: A new approach to error arithmetic, SIAM J. Numer. Anal. 15 (1978), 368-393.

33. Rall, L. B. (Ed.): Error in Digital Computation, Vol. 1. New York: John Wiley & Sons, 1965.

34. Rall, L. B. (Ed.): Error in Digital Computation, Vol. 2. New York: John Wiley & Sons, 1965.

35. Rall, L. B.: Computational Solution of Nonlinear Operator Equations. New York: John Wiley & Sons, 1969.

36. Rall, L. B.: A comparison of the existence theorems of Kantorovich and Moore, Tech. Sum. Rpt. No. 1944, Math. Res. Center, Univ. Wisconsin-Madison, March, 1979.

37. Rall, L. B. and Wanner, G.: Experience with Lie series, Meth. und Verfahren der Math. Physik 5 (1971), 29-42.

38. Reiter, Alan: Automatic generation of Taylor coefficients (TAYLOR), Prog. No. 3, Math. Res. Center, Univ. Wisconsin-Madison, July, 1965.

39. Reiter, Alan: Interval arithmetic package (INTERVAL), Prog. No. 2, Math. Res. Center, Univ. Wisconsin-Madison, June, 1965.

40. Reiter, Alan: Compiler of differential expressions (CODEX), Prog. No. 1, Math. Res. Center, Univ. Wisconsin-Madison, Aug., 1965.

41. Reiter, Allen: Automatic generation of Taylor coefficients (TAYLOR) for the CDC 1604, Tech. Sum. Rpt. No. 830, Math. Res. Center, Univ. Wisconsin-Madison, Nov., 1967.

42. Reiter, Allen: Interval arithmetic package (INTERVAL) for the CDC 1604 and CDC 3600, Tech. Sum. Rpt. No. 794, Math. Res. Center, Univ. Wisconsin-Madison, Jan., 1979.

43. Stirling, James: Methodus Differentialis: sive Tractatus de Summatione et Interpolatione Serierum Infinitarum. London: Typis, Gul. Bowyer, Impensis, G. Strahan, 1730.

44. Stummel, F.: Rounding error analysis of numerical algorithms, to appear.

45. Wertz, H. J.: SUPER-CODEX (Supervisor plus a compiler of differentiable expressions, Math. Res. Center, Univ. Wisconsin-Madison, June, 1968.

46. Wertz, H. J.: SUPER-CODEX: Analytic differentiation of FORTRAN statements, Rpt. No. TOR-0172(9320)-12, Aerospace Corporation, El Segundo, Calif., April, 1972.

47. Yohe, J. M.: The interval arithmetic package, Tech. Sum. Rpt. No. 1755, Math. Res. Center, Univ. Wisconsin-Madison, June, 1977.

48. Yohe, J. M.: Implementing nonstandard arithmetics, SIAM Rev. 21 (1979), 34-56.

49. Yohe, J. M.: Portable software for interval arithmetic, to appear.

LBR/scr

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 1976	2. GOVT ACCESSION NO. (9)	3. RECIPIENT'S CATALOG NUMBER Technical
4. TITLE (and Subtitle) (6) APPLICATIONS OF SOFTWARE FOR AUTOMATIC DIFFERENTIATION IN NUMERICAL COMPUTATION.		5. TYPE OF REPORT & PERIOD COVERED Summary Report - no specific reporting period
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) L. B. Rall		8. CONTRACT OR GRANT NUMBER(s) (15) DAAG29-75-C-0024
9. PERFORMING ORGANIZATION NAME AND ADDRESS Mathematics Research Center, University of Wisconsin 610 Walnut Street Madison, Wisconsin 53706		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 7 (Numerical Analysis) 8 (Computer Science)
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office P.O. Box 12211 Research Triangle Park, North Carolina 27709		12. REPORT DATE (11) Jul 1979
		13. NUMBER OF PAGES 22
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 27		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(14) MRC-TSR-1976

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Automatic differentiation
Taylor coefficient generation
Error estimation
Differential equations
Nonlinear systems
Optimization
Numerical integration
Interval analysis

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

At present, software for formula translation is used routinely in numerical computation. On the other hand, although software for differentiation of formulas is easy to construct and widely available, many numerical analysts seem to be unaware of its existence and potential for numerical computation. A simple procedure for formula translation and differentiation will be described, and some significant applications will be indicated. In ordinary computation, these include solution of ordinary and partial differential equations by series methods (Taylor and Lie series, for example),

221 200

20. ABSTRACT - Cont'd.

solution of nonlinear systems of equations by Newton's method and its variants, and nonlinear optimization (constrained and unconstrained). Together with interval analysis, differentiation can be used to determine properties of functions and thus automate the application of certain theorems, such as ones for existence of fixed points of n-dimensional operators or solutions of nonlinear systems of equations. Another large field of application of differentiation is to automatic error analysis, either using the graph structure of the computation, or interval analysis. An example is given of a program for numerical integration in which automatic differentiation and interval analysis are combined to provide <u>a priori</u> and <u>a posteriori</u> error bounds for the results.